HASHING

Compiled By
Ms. Shruthi M
Dept of CSE.

- Time required to access any element in an array irrespective of its position is same.

- Physical location of i^{th} item in array is calculated by multiplying the size of each element of array with i and adding to base address:

$$Loc(A_i) = Base\text{-}address + i * c$$

i is the index, c is size of each element of array.

- Address of any item can be obtained and from the address obtained we can access the item.

- Similar idea is used in hashing to store and retrieve the data.

Definition:

- Data can be stored in the form of a table using arrays and from this table data can be retrieved.
- The table on which insertion, deletion and retrieve operations takes place is called hash table.
- Hash table can be implemented using an ordinary array H[0 .. m-1].
- Hash table size is limited and so it is necessary to map given data into fairly restricted set of integers.
- Process of mapping large amount of data into a smaller table is called hashing.
- Hash function provides mapping between original data and smaller table by transforming a key into an index to table is called hash table.

- This key that transforms a key into index to hash table is called hash function.
- Various keys are distributed among locations 0 to m-1 and it is required to compute the keys using hash function.
- Hash function assigns an integer value from 0 to m-1 to keys and values act as index to hash table are called hash addresses or hash values.
- Hash function should be such that all keys are distributed as evenly as possible among various cells of table.
- Computation of key by hash function should be simple.

Collision and its detection

- If size of hash table is fixed and if size of table is smaller than total number of keys generated , then two or more keys will have same hash address.
- The phenomenon of two or more keys hashed in same location of hash table is called collision.

- Collision can also occur even if number of keys are less than size of hash table.

- All keys if have same value results in worst case collision. Such case all keys are stored in one cell in the form of list.
- Search will be O(n) times.

Hashing techniques

To avoid collision various hashing techniques are used:

1. Open hashing (separate chaining)
2. Closed hashing (open addressing)

Open Hashing:

- Collisions can be avoided using this very simple method. Can be achieved using an array of linked lists.
- If an item is to be inserted using a hash function, first hash address is obtained.
- Hash address is used to find the list to which the item is to be added and using appropriate function item can be inserted at the end of the list.
- If more than one element has same hashing addresses, all keys are hashed into same address will be inserted at the end of list and hence collision is avoided.

- Searching using this technique takes less time.
- While searching find hash address using hash function.
- Hash address corresponds to an index obtains address of list, if address is NULL, item is not present, If not NULL list is traversed sequentially to search for item. If item is found search is successful else unsuccessful.

- Deleting a node, hash function determines address acting as index of item to be deleted. Linked list corresponding to index is searched and if item is present then deleted.
- Else report as not found

- Only disadvantage: requires extra storage space for pointers coz of linked lists.

the index is searched and if the item is present, the corresponding node in that list can be deleted. The linked list corresponding to be deleted. During searching if end of list is encountered, the item not found and report that it is not possible to delete as the item not found.

The only disadvantage of this technique is that linked list requires extra storage space for the pointers because of linked lists.

Words	Sum of positions	hash address = sum %5
LIKE	12+9+11+5 =37	2
A	1 =1	1
TREE	20+18+5+5 =48	3
YOU	25+15+21 =61	1
FIRST	6+9+18+19+20 =72	2
FIND	6+9+14+4 =33	3
A	1 =1	1
PLACE	16+12+1+3+5 = 37	2
TO	20+15 =35	0
GROW	7+18+15+23 =63	3
AND	1+14+4 = 19	4
THEN	20+8+5+14 =47	2
BRANCH	2+18+1+14+3+8 =46	1
OUT	15+21+20 =56	1

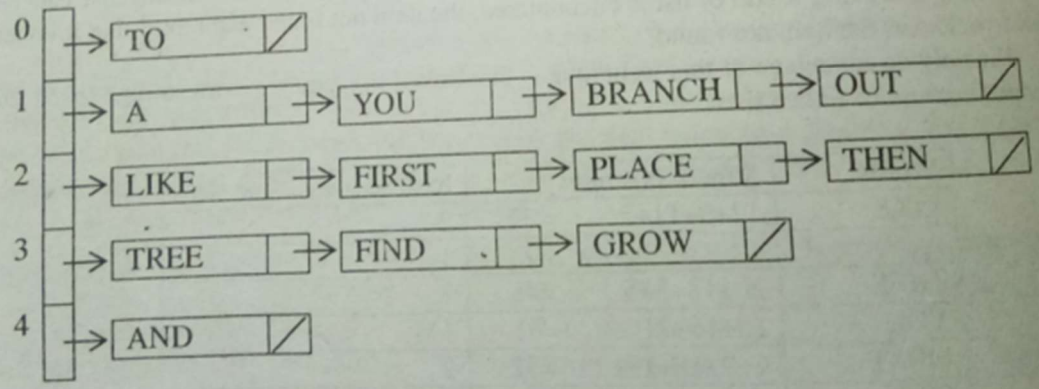
Table 7.4 To compute hash addresses

Example 7.4.1: Let the size of hash table is 5. Insert the following words LIKE, A, TREE, YOU, FIRST, FIND, A, PLACE, TO, GROW, AND, THEN, BRANCH, OUT into a hash table using appropriate hash function and show how each item is inserted into the table, how an item can be searched and deleted

Solution: A hash table of size 5 indicates that it is required to construct a hash table by using an array of 5 linked lists. Let us discuss the way to create a hash function, to insert an element into the table, to search for an element and to delete an element from the table.

Design of Hashing function: Let us design a simple hashing function which will accept a string consisting of a key and add the positions of word's letters in the alphabet and compute the remainder by dividing the sum by the size of the table. The words to be inserted into the hash table, the sum of positions of each letter in the word and the hash address are shown in the table 7.4. The size of the hash table is assumed to be 5 and so while taking the remainder divide the sum by 5 so that all the hash address of all the words lie within 0 and 4.

Construction of Hash table: Now, let us see how to construct a hash table. Since the size of the hash table is 5, we will have an array of 5 rows with each row having a linked list. Obtain the words one by one, find the hash address and insert at the end of the list in the appropriate location in the array. The hash table obtained by computing the hash addresses(see table 7.4) is shown below:



Observe from the table 7.4 that hash value of word 'TO' is 0 (see table 7.4). So, it is inserted into h[0]. The hash value of the words 'LIKE', 'FIRST', 'PLACE' and 'THEN' is 2 and the words are inserted into h[2] at the end of the list one by one as and when they are scanned. Thus, a hash table can be created.

Search for the given pattern: Searching is very efficient if the hash table size is appropriately selected and the items are distributed evenly in the table. If it is required to search for the string *str*, then compute the hash address (hash value) of the string *str*. If there is no entry in the location identified by hash address, string *str* is not present in the table and search is unsuccessful. If there is any entry in that location, search for the string in the corresponding list and if found it can be retrieved. For example, if the word to be searched is "RAMA", Find the hash value which is given by (18+1+13+1) % 5 = 33 % 5 = 3. So, it is required to search for "RAMA" in h[3]. In the list identified by h[3] has the words "TREE", "FIND" and "GROW". The string "RAMA" is compared with all the words in the list. Since there is no matching word, unsuccessful message has to be displayed and if found, display the appropriate message. Thus, string search can be achieved very efficiently using *hashing technique*.

Deletion of specified word: Like search operation, deletion of specified item is also very efficient if the hash table size is appropriately selected and the items are distributed evenly in the table. If it is required to delete for the string *str*, then compute the hash address (hash value) of the string *str*. If there is no entry in the location identified by hash address, string *str* is not present in the table and search is unsuccessful and so deletion is not possible. If there is any entry in that location, search for the string in the corresponding list and if found the corresponding node can be deleted from that list.

Closed Hashing (open addressing):

- Amount of space available for storing various data is fixed at compile time by declaring a fixed array for the hash table, so all keys are stored in this fixed hash table itself without the use of linked lists.

- Collision is avoided here by finding another, unoccupied location in the array.

Strategies used to avoid collision are:

- Linear probing
- Double hashing

Linear Probing:

- If an item is inserted, as usual, using the hash function find the hash address which gives an index into the array.
- If the position identified by the index is empty, an item is inserted at that location.
- In case at that position if an item already exists, then a collision occurs and a next empty location has to be searched and item has to be inserted at the empty location.

Example:

If collision occurs at i^{th} position, from i + 1 onwards are verified for empty locations and whenever we get an empty location, item is inserted at that position.

To search an item, find hash address using hash function. If location identified by h(key) is empty, item is not found search is

Unsuccessful.

- check the item with items in following locations until the item is found or an empty location is encountered, and hence search is unsuccessful.

example 7.4.2 with a typical

Example 7.4.2: Let the size of hash table is 15. Insert the following words LIKE, A, TREE, YOU, FIRST, FIND, A, PLACE, TO, GROW, AND, THEN, BRANCH, OUT into a hash table using appropriate hash function and show how each item is inserted into the table, how an item can be searched and deleted using linear probing

Note: In closed hashing, the size of the hash table must be at least equal to the number of unique words in a given text. For example, in the string "LIKE A TREE YOU MUST FIND A PLACE TO GROW AND THEN BRANCH OUT", even though the number of words are 14, the number of unique words is 13 (The word A appears twice). So, the size of the hash table must be at least 13 and it can be larger than 13. Let us assume the table size is 15.

Insert an item: The words to be inserted into the hash table are: LIKE, A, TREE, YOU, FIRST, FIND, A, PLACE, TO, GROW, AND, THEN, BRANCH, OUT. As usual find the hash address by taking the sum of positions of alphabets in the word and taking the remainder by dividing it by 15. The various hash addresses are shown below:

Sl. No	Words	Sum of positions		hash address = sum %13
1.	LIKE	12+9+11+5	=37	7
2.	A	1	=1	1
3.	TREE	20+18+5+5	=48	3
4.	YOU	25+15+21	=61	1
5.	FIRST	6+9+18+19+20	=72	12
6.	FIND	6+9+14+4	=33	3
7.	A	1	=1	1
8.	PLACE	16+12+1+3+5	= 37	7
9.	TO	20+15	=35	5
10.	GROW	7+18+15+23	=63	3
11.	AND	1+14+4	= 19	4
12.	THEN	20+8+5+14	=47	2
13.	BRANCH	2+18+1+14+3+8	=46	1
14.	OUT	15+21+20	=56	11

Table 7.5 To compute hash addresses

The words are taken in the order of serial number from the table 7.5 and inserted into the hash table one by one based on the hash address(value or index). The words "LIKE", "A" and "TREE"

are inserted into positions 7, 1 and 3 respectively as shown in table 7.6.a. The next word to be selected is "YOU" whose hash value is 1 and is already full. So, it is inserted in the next available location 2 as shown in table 7.6.b. But, in location 1 of table 7.6.a, an entry of "TREE" already exists. So, the next available empty location is 10 and so, the word "YOU" is inserted at position 10 as shown in table 7.6.b. The word "FIRST" whose hash value is 12 is inserted at position 12 since the location is empty. The word "FIND" whose hash value is 3, is inserted at position 4, since position 3 is not empty. The word "A" whose value is 1 is not considered since it is already available in position 1. The word "PLACE" whose hash value is 7 is inserted at position 8, since the location 7 is not empty. The word "TO" has hash value of 5 and the location is empty (see table 7.6.b.) and so it is inserted at position 5 as shown in table 7.6.c. The word "GROW" has the hash value 3 and since the location is full, the next available location is 6 and so it is inserted at position 6. Similarly, the word "AND", "THEN", "BRANCH" and "OUT" are inserted at positions 9, 10,11 and 13 because the corresponding locations are full.

0	
1	A
2	
3	TREE
4	
5	
6	
7	LIKE
8	
9	
10	
11	
12	
13	
14	
15	

0	
1	A
2	YOU
3	TREE
4	FIND
5	
6	
7	LIKE
8	PLACE
9	
10	
11	
12	FIRST
13	
14	
15	

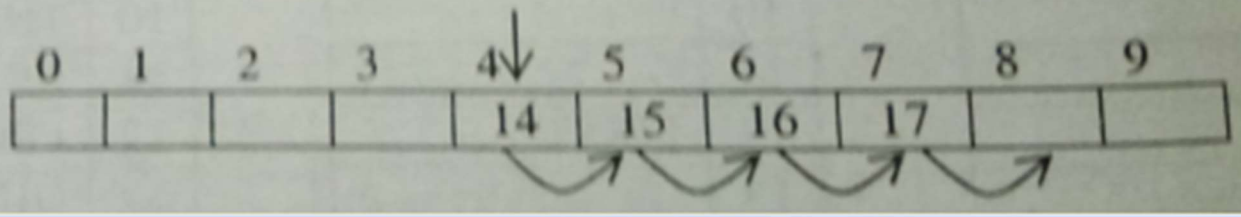
0	
1	A
2	YOU
3	TREE
4	FIND
5	TO
6	GROW
7	LIKE
8	PLACE
9	AND
10	THEN
11	BRANCH
12	FIRST
13	OUT
14	
15	

Search for an item Consider the word "KID" to be searched. The hash value k(KID) = (11 + 9 + 4) % 15 = 9. So, compare the word "KID" with the items as position 9 to 13 and is not available in those locations and location 14 is empty and so, the search stops and we display unsuccessful search. If the word to be searched is "BRANCH" whose hash value is 1, it is required to search for that word from position onwards, till the word found or till an empty location is found. Since, it is available in location 11, report that the search is successful.

Note: During searching using this technique, we assume that the array is circular, so that if we search past the end of the hash table, we start again at item 0.

Delete an item: Deleting an element from the hash table using this technique is quite complex. Suppose, it is required to delete the word "TO". Its hash value is 5. So, the location 5 is searched for the word "TO" and since there is a match, we can delete that word from that location. But, we can not delete the word "BRANCH". Because, the hash value of the word "BRANCH" is 1. The algorithm searches for the word "BRANCH" from position 1 onwards. Since the word "TO" at location is already deleted, the algorithm encounters an empty location and report the unsuccessful search result. This problem can be avoided by a technique called "lazy deletion", that is, mark the previously occupied locations by a special symbol to distinguish them from the locations that have not been occupied. So, the complexity of the algorithm increases.

Disadvantages: The disadvantage of linear probing is that its performance deteriorates because of the phenomenon called *clustering*. ***Clustering*** is a phenomenon where the data tend to cluster (gather) at certain points in the table, with other part of the table not being used. This phenomenon leads to lengthy sequential searches through the table when attempting to retrieve the data. For example, even though hash value of the word "BRANCH" is 1, since earlier items are already clustered together, there was no empty location nearby and so it was inserted at location 11 and searching time will naturally increase. Thus, once the data are filled in nearby locations, if another item has to be inserted which has the same hash value, a lengthy search has to be made for empty location and then the item has to be inserted. Thus inserting subsequent elements leads to longer and longer clusters. For example, consider the figure 7.4.2 which shows an item being inserted into a hash table using linear probing to resolve collision. Assume four elements are inserted already into a table. Now, suppose a new item to be added has the hash value of 4. Since the locations 4, 5, 6 and 7 are already full, the collision resolution technique eventually places the item in location 8.



Primary Clustering Whenever an insertion is made between two clusters that are separated by an empty location, the two clusters become one, thereby potentially increasing the cluster length. This phenomenon is called primary clustering. For example, there are two clusters: the first cluster has three elements 11, 12 and 13 and the second cluster has four elements 15, 16, 17 and 18. If an item say 14 is inserted at position 4, the insertion results in a single clustering consisting of elements from 11 to 18.

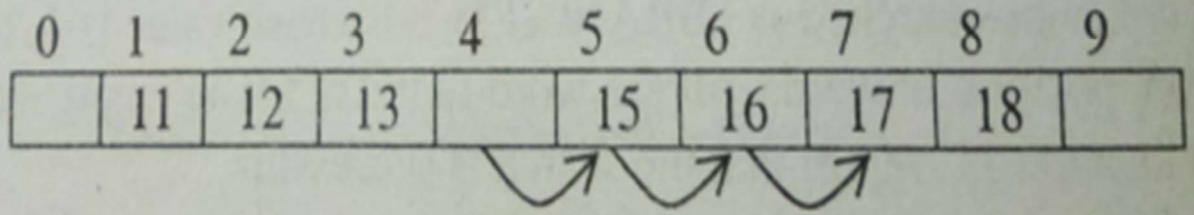


Figure 7.4.3 Primary clustering

Double hashing:

- Uses two distinct hash functions h(key) and s(key) both functions accept same key as parameter.
- Whenever item has to be retrieved or inserted, primary hashing function i = h(key) is used to determine the position.
- Table size m is used in primary hashing function.
- If that position is occupied by some other element , secondary hash function s(key) is called

  i = (i + s(key)) % m-1 // m is size of hash table.